Applying Runtime Verification Techniques to Enterprise Service Bus Architectures

Christian Colombo[1], Gabriel Dimech[1], and Adrian Francalanza[1]

Department of Computer Science, University of Malta
{christian.colombo | gdim0002 | adrian.francalanza}@um.edu.mt

The increased usage of distributed systems has led to the problem of integration of IT systems that communicate via different protocols. In such setting, it is also typical for these components to be added/removed from the system at runtime, depending on requirements. These two characteristics give rise to integration challenges where systems should be able to communicate seamlessly with each other whilst being able to be easily integrated with an existing distributed system with minimal impact.

An ESB (Enterprise Service Bus) is a tool which attempts to address the above issues by integrating systems in a bus-like architecture where a central bus enables components to communicate via messages [5, 2]. This arrangement, transparently handling complex messaging functions such as transformation and routing (via routing patterns), enables the user to focus on the business logic, abstracting away communication concerns [6].

Despite facilitating orchestration of distributed components in a scalable manner, current ESBs provide little support for correctness guarantees of the overall system logic e.g. a booking component may only confirm the booking once the bank component has verified the payment details, and the airline component confirms that the dates specified are permissible.

Popular techniques for ensuring correctness, such as testing and model checking are not ideal for verifying the correctness of ESB systems due to the latter's highly dynamic nature. For this reason, we propose to apply runtime verification [4, 3, 1] techniques, promising a scalable approach to checking all observed behaviour under any runtime circumstance — no matter how unpredictable this may be.

Design Options for a Runtime Verification Approach

There are a number of concerns which have to be considered when applying runtime verification techniques to ESBs: (*i*) **Dynamic updating of network** Due to the dynamic nature of ESBs, monitors should be able to tolerate a network of components which join and leave the network at will. (*ii*) **Expressivity** The formalism used to express the logic to be runtime verified should support the encoding of typical logic found in ESBs. (*iii*) **Execution efficiency** The verification code added to the ESB framework should not introduce prohibitive execution overheads. (*iv*) **Communication efficiency** Communication between

the potentially numerous components of the runtime verifier should not interfere with the rest of the messages. (v) **Privacy issues** The verification process should not lead to exposure of any private information across ESB components.

While there are numerous points on the design space of runtime verification applications to ESBs, in this short overview we focus on two main ones: the orchestration and the choreography approach. In the former, the verification of properties is done via a central monitor which is able to observe all the communication channels of the distributed system. Orchestration-based approaches are relatively straight forward to design and implement, as the verifier design does not require communicating with other verifiers. However, a disadvantage of orchestration-based approaches, is that since the monitor is in one central location, monitoring performance impact directly affects the ESB. Additionally, the network formed between components and the ESB is also impacted as information required for verification must be communicated over message channels. Figure 1 depicts an orchestration-based verification setup where three components A, B, and C are communicating events to a central monitor.

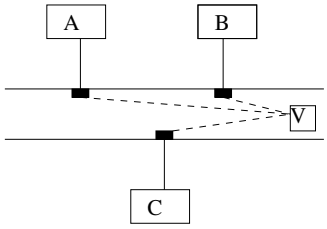

Fig. 1. Orchestration-based monitoring

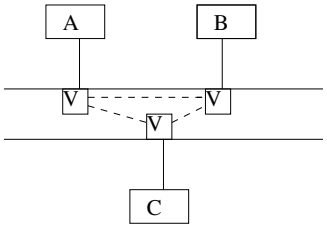

Fig. 2. Choreograph-based monitoring

Choreography-based verification involves dividing the verification code in separate sub verifiers designed to communicate with one another so that they are able to verify a property. One variant of choreography-based verification is to push the monitoring of properties at runtime onto the components forming the distributed system. In doing this, performance impact of verification on the ESB is decreased as the sub-verifiers on the components shall be performing most of the verification, only communicating with other verifiers when required. Performance impact on the network is also lessened due to the fact that the central verifier residing on the ESB now requires less information from the components for monitoring purposes. In addition, these monitors are able to verify local properties for the respective components in isolation from monitors residing on other connected components. However this approach requires that the remote components both allow and trust the local monitors to execute on their behalf. Distributing verifiers on remote components is usually only possible in a setting where the distributed system is controlled by a single organisation. One other variant of choreography-based verification is to apply sub verifiers on the message channels residing on the ESB rather than on the remote components. On

the one hand, this has the disadvantage of pushing the overhead onto the ESB infrastructure as in the case of the orchestration-based approach. On the other hand, having the verifier module split into sub-verifiers enables us to dynamically switch on and off parts of the verification network to keep the overheads to a minimum. Figure 2 shows a depiction of a choreograph-based verification setup.

Conclusion

The discussion presented in this short abstract does not cover all the criteria outlined in the previous section but gives an introduction to the issues involved in choosing the right design for a runtime verification setup. In the future, we shall be considering various design options in the context of real-life ESB applications with the aim of finding the right tradeoffs between expressivity, overheads, and correctness assurance.

References

1. Howard Barringer, Yliès Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon J. Pace, Grigore Rosu, Oleg Sokolsky, and Nikolai Tillmann, editors. *RV*, volume 6418 of *LNCS*, 2010.
2. David A. Chappell. *Enterprise Service Bus: Theory in Practice*. O'Reilly, 2004.
3. Séverine Colin and Leonardo Mariani. Run-time verification. In *Model-Based Testing of Reactive Systems*, volume 3472 of *Lecture Notes in Computer Science*, pages 525–555, 2004.
4. Martin Leucker and Christian Schallhart. A brief account of runtime verification. *JLAP*, 78(5):293–303, 2009.
5. Anne Thomas Mane. Enterprise service bus: A definition, May 2013.
6. Ross Mason. Mediation - separating business logic from messaging, May 2013.